Problem Solving Notes

None

Jaemin Park

Table of contents

1. Home

2. 문제 풀이

2.1 문제 풀이

2.2 Olympiad

2.2.1 JOISC

2023

binary_search data_structures euler_tour_technique lca offline_queries pbs pst segtree trees ~ solution

JOISC 2023 P1.1 TWO CURRENCIES

문제

문제 링크

https://www.acmicpc.net/problem/27992
https://oj.uz/problem/view/JOI23_currencies

문제 요약

정점 N개의 트리가 주어지고, 몇 개의 간선은 지나려면 통행료를 내야 한다.
P_i번째 간선을 통과하기 위해서는 1개의 금화를 내거나, C_i개의 은화를 내야 한다. $(1 \le i \le M)$
Q개의 쿼리가 주어지며, i번 쿼리에서는 S_i번 정점에서 T_i번 정점으로 금화 X_i개, 은화 Y_i개를 갖고 이동하려 한다. $(1 \le i \le Q)$
이 때, 이동이 가능한지 판별하고, 가능하다면 남는 금화의 개수의 최댓값을 구하여라.

제한

- $2 \le N \le 100{,}000$
- $1 \le M \le 100{,}000$
- $1 \le Q \le 100{,}000$
- $1 \le A_i, B_i \le N$ $(1 \le i \le N-1)$
- $1 \le P_j \le N-1$, $1 \le C_j \le 10^9$ $(1 \le j \le M)$
- $1 \le S_k, T_k \le N$, $0 \le X_k \le 10^9$, $0 \le Y_k \le 10^{18}$ $(1 \le i \le Q)$

입력 / 출력

> Input

$N\ M\ Q$
$A_1\ B_1$
$A_2\ B_2$
⋮
$A_{N-1}\ B_{N-1}$
$P_1\ C_1$
$P_2\ C_2$
⋮
$P_M\ C_M$
$S_1\ T_1\ X_1\ Y_1$
$S_2\ T_2\ X_2\ Y_2$
⋮
$S_Q\ T_Q\ X_Q\ Y_Q$

- A_i, B_i는 트리의 인접 리스트
- P_i는 간선 번호

> Output

ans_1
ans_2
⋮
ans_Q

- ans_i는 i번째 쿼리에 대하여 불가능하면 -1, 가능하다면 남는 금화의 개수의 최댓값

풀이

Subtask 1

- $N \le 2000$, $M \le 2000$, $Q \le 2000$

한 쿼리에서 이동하는 경로에 있는 모든 통행료들을 모아서 생각하자. 통행료들의 배열 C_i는 금화 1개를 내거나, 은화 C_i개를 내야 함을 의미한다. 사용할 수 있는 은화의 총 개수가 정해진 상태에서, 금화를 최대한 적게 사용하는 것이 목적이니, C_i가 작은 통행료부터 예산이 바닥날 때까지 은화로 지불하고, 나머지를 금화로 지불하는 것이 최선이다.

> Observation 1

한 쿼리 k에서 경로에 있는 모든 통행료들을 C_i의 순서대로 정렬한 후, C_i가 작은 통행료부터 순서대로 합이 Y_k이하일 때까지 최대한 더하고 남은 통행료들을 금화로 지불하는 것이 최적해이다.

Observation 1을 이용하여 Naive하게 각 쿼리마다 경로를 따라 모든 통행료들의 배열을 얻고, 정렬하여 답을 계산하면 $O(Q(N+M))$에 문제를 해결할 수 있다.

> CheckPoint

Observation 1을 이용하여 Naive하게 구현하면 $O(Q(N+M))$에 문제를 해결할 수 있다.

Complexity

Time Complexity : $O(Q(N+M))$

Subtask 4 (Full)

N, Q, M이 클 때에는 직접 C_i들을 정렬하여 답을 구할 수 없다. 직접 정렬하지 않고, Observation 1의, C_i가 작은 값들부터 이용해야 한다는 조건을 어떻게 활용하면 좋을까?

이분탐색을 통해 해결할 수 있다. 우선, 편의를 위하여 모든 C_i가 서로 다르다고 가정한다.
$C_i \le T$인 통행료들의 합을 구하여 통행료들의 합이 Y_k보다 작거나 같은 T의 최댓값을 구하면, $C_i \le T$인 통행료들의 개수가 곧 은화로 지불할 수 있는 통행료들의 수를 의미한다. 이제 경로 (S_k,T_k)에 있는 통행료들 중 T 이하인 것들의 합, 혹은 개수에 대한 쿼리에 빠른 시간 안에 답할 수 있다면, 이 결과를 이용하여 이분탐색을 통해 답을 구할 수 있다.

CheckPoint

경로 (S_k,T_k)에 있는 통행료들 중 T 이하인 것들의 합, 혹은 개수에 대한 쿼리에 빠른 시간 안에 답할 수 있다고 가정하자. $C_i \le T$인 통행료들의 합이 Y_k보다 작거나 같은 T의 최댓값을 이분탐색으로 구하여 문제를 해결할 수 있다.

루트에서 임의의 정점까지의 경로에서 T이하인 수들의 개수나 합은 Persistent Segment Tree를 활용하여 구할 수 있다. 트리에서 DFS를 돌며, 자식의 PST는 부모의 PST에 간선이 지나는 C_i를 추가한 형태로 PST를 구성한 후, 경로 (S_k,T_k)의 쿼리는 S_k, T_k, lca 에서의 T에 대한 쿼리를 날려 구할 수 있다. 이분탐색에 $O(logY)$, PST 쿼리에서 $O(logM)$의 시간이 걸리니, 전체 $O(NlogN+M+QlogYlogM)$에 문제를 해결할 수 있다.

CheckPoint

필요한 쿼리는 트리에서의 Persistent Segment Tree를 이용하여 구할 수 있다.
전체 시간복잡도는 $O(NlogN+M+QlogYlogM)$이다.

Complexity

Time Complexity : $O(NlogN+M+QlogYlogM)$

모든 쿼리가 오프라인임을 이용하면 상수가 큰 PST를 사용하지 않고도 문제를 해결할 수 있다. Q번의 이분탐색을 해야 하니, Parallel Binary Search를 사용하자.

모든 쿼리에 대하여 동시에 이분탐색을 하고, 한 번 이분탐색을 할 때 T와 C_i들을 정렬한다. 정렬된 순서대로 트리에 간선 가중치 업데이트, 경로의 가중치 합을 구할 수 있어야 한다. 이는 Euler Tree Trick을 사용해서 트리를 일직선으로 핀 후, 구간 업데이트, 점 쿼리로 해결할 수 있으니 Segment Tree (Lazy Propagation 필요 없음)으로 구한다. 이분탐색에 총 $O(logY)$, 정렬 및 Segment Tree 쿼리에 $O(logN)$의 시간이 걸리니, 전체 $O(NlogN+M+QlogYlogN)$에 문제를 해결할 수 있다.

CheckPoint

오프라인 쿼리이니, Parallel Binary Search를 사용하면 간선 가중치 업데이트와 경로 가중치 합 쿼리 문제로 환원할 수 있다. 이는 Euler Tree Trick으로 트리를 일직선으로 피고, Segment Tree를 활용하여 해결할 수 있다.
전체 시간복잡도는 $O(NlogN+M+QlogYlogN)$이다.

Complexity

Time Complexity : $O(NlogN+M+QlogYlogN)$

마지막으로, 모든 C_i가 다르다고 가정하였는데, C_i가 같을 수 있다면 같은 C_i중 일부만 사용하는 경우를 고려할 수 없다. 이는 단순히 임의로 `tie-breaking rule`을 적용한 후, 이분탐색을 정렬된 C_i들의 배열에 대해서 시행함으로서 해결할 수 있다.

코드

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

const int MAXN = 1e5;

int N, M, Q;
vector<pii> adj[MAXN+10];
pii A[MAXN+10];

struct SEG
{
    ll tree[MAXN*4+10];
    void init()
    {
        fill(tree, tree+MAXN*4+10, 0);
    }
    void update(int node, int tl, int tr, int l, int r, ll k)
    {
        if(l<=tl && tr<=r)
        {
            tree[node]+=k;
            return;
        }
        if(r<tl || tr<l) return;
        int mid=tl+tr>>1;
        update(node*2, tl, mid, l, r, k);
        update(node*2+1, mid+1, tr, l, r, k);
    }
    ll query(int node, int tl, int tr, int k)
    {
        if(tl==tr) return tree[node];
        int mid=tl+tr>>1;
        if(k<=mid) return tree[node]+query(node*2, tl, mid ,k);
        else return tree[node]+query(node*2+1, mid+1, tr, k);
    }
}seg;

int L[MAXN+10], R[MAXN+10], dep[MAXN+10], par[MAXN+10][30], cnt, P[MAXN+10];
void dfs(int now, int bef, int d)
{
    L[now]=++cnt;
    par[now][0]=bef;
    dep[now]=d;
    for(auto [nxt, e] : adj[now])
    {
        if(nxt==bef) continue;
        P[e]=nxt;
        dfs(nxt, now, d+1);
    }
    R[now]=cnt;
}

int lca(int u, int v)
{
    if(dep[u]>dep[v]) swap(u, v);
    for(int i=20; i>=0; i--) if(dep[par[v][i]]>=dep[u]) v=par[v][i];
    if(u==v) return u;
    for(int i=20; i>=0; i--) if(par[u][i]!=par[v][i]) u=par[u][i], v=par[v][i];
    return par[u][0];
}

int C[MAXN+10];
pii B[MAXN+10];
pll D[MAXN+10];
int lo[MAXN+10], hi[MAXN+10];
int E[MAXN+10];

ll f(int p)
{
    int u=B[p].first, v=B[p].second, w=C[p];
    return seg.query(1, 1, N, L[u])+seg.query(1, 1, N, L[v])-2*seg.query(1, 1, N, L[w]);
}

int main()
{
    scanf("%d%d%d", &N, &M, &Q);
    for(int i=1; i<N; i++)
    {
        int u, v;
        scanf("%d%d", &u, &v);
        adj[u].push_back({v, i});
        adj[v].push_back({u, i});
    }
    for(int i=1; i<=M; i++) scanf("%d%d", &A[i].second, &A[i].first);

    dfs(1, 1, 1);
    for(int i=1; i<=20; i++) for(int j=1; j<=N; j++) par[j][i]=par[par[j][i-1]][i-1];
```

```
    for(int i=1; i<=Q; i++) scanf("%d%d%lld%lld", &B[i].first, &B[i].second, &D[i].first, &D[i].second), C[i]=lca(B[i].first, B[i].second);

    sort(A+1, A+M+1);

    for(int i=1; i<=Q; i++) lo[i]=0, hi[i]=M+1;
    while(1)
    {
        vector<pii> V;
        for(int i=1; i<=Q; i++)
        {
            if(lo[i]+1<hi[i])
            {
                V.push_back({lo[i]+hi[i]>>1, i});
            }
        }
        if(V.empty()) break;
        sort(V.begin(), V.end());

        seg.init();
        int pt=1;
        for(auto it : V)
        {
            for(; pt<=M && pt<=it.first; pt++)
            {
                seg.update(1, 1, N, L[P[A[pt].second]], R[P[A[pt].second]], A[pt].first);
            }
            //printf("??%d %d %lld %d\n", it.first, it.second, f(it.second), D[it.second].second);
            if(f(it.second)>D[it.second].second) hi[it.second]=it.first;
            else lo[it.second]=it.first;
        }
    }

    seg.init();
    int pt=1;
    vector<pii> V;
    for(int i=1; i<=Q; i++) V.push_back({lo[i], i});
    sort(V.begin(), V.end());
    for(auto it : V)
    {
        for(; pt<=M && pt<=it.first; pt++)
        {
            seg.update(1, 1, N, L[P[A[pt].second]], R[P[A[pt].second]], 1);
        }
        E[it.second]=f(it.second);
    }
    for(; pt<=M; pt++) seg.update(1, 1, N, L[P[A[pt].second]], R[P[A[pt].second]], 1);

    for(int i=1; i<=Q; i++)
    {
        //printf("%d %d %d\n", lo[i], f(i), E[i]);
        printf("%lld\n", max(-1ll, D[i].first-f(i)+E[i]));
    }
}
```

0_1_bfs data_structures dijkstra graphs segtree ~ solution

JOISC 2023 P1.3 PASSPORT

문제

문제 링크

https://www.acmicpc.net/problem/27994
https://oj.uz/problem/view/JOI23_passport

문제 요약

1,2,⋯ ,N번의 N개의 도시가 있고, 도시 i에서는 구간 $[L_i, R_i]$내의 임의의 도시로 자유롭게 이동할 수 있는 여권을 발급받을 수 있다. $(1 \le i \le N)$
Q개의 쿼리가 주어질 때, 각 쿼리별로 X_k번 도시에서 출발하였을 때, 모든 도시를 방문하기 위하여 필요한 여권의 최소 개수를 구하여라. $(1 \le k \le Q)$

제한

- $2 \le N \le 200{,}000$
- $1 \le L_i \le i \le R_i \le N \ (1 \le i \le N)$
- $1 \le Q \le N$
- $1 \le X_k \le N \ (1 \le k \le Q)$, 모든 X_k는 서로 다르다.

입력 / 출력

> Input

N
$L_1 \ R_1$
$L_2 \ R_2$
⋮
$L_N \ R_N$
Q
X_1
X_2
⋮
X_Q

> Output

ans_1
ans_2
⋮
ans_Q

풀이

Subtask 4

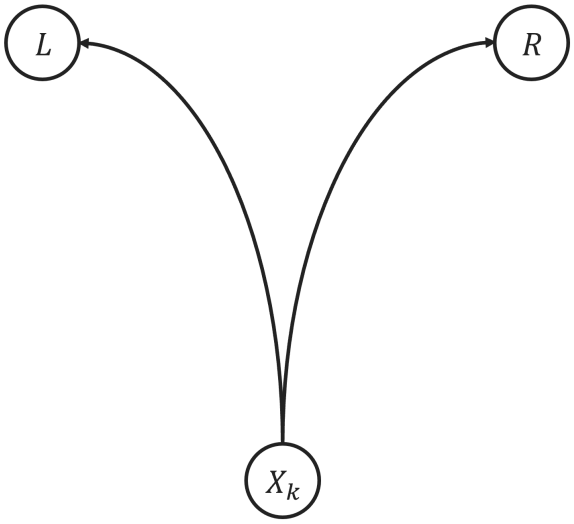
- $N \le 2500$

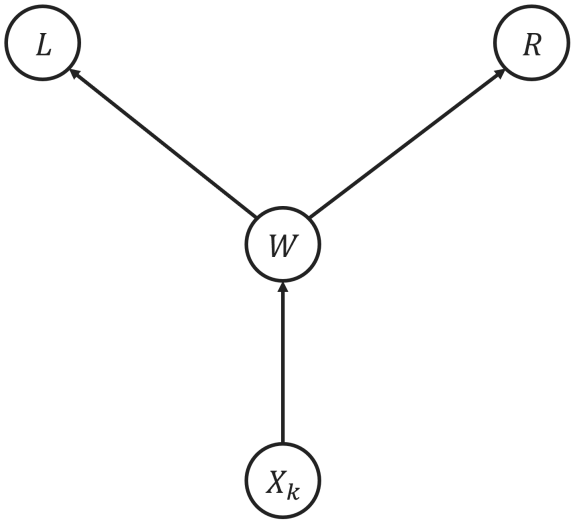
현재 갖고 있는 여권들만 사용해서 이동할 수 있는 도시의 범위는 항상 구간이다. $(\because L_i \le i \le R_i)$
따라서, 방문 가능한 도시의 범위가 $[1, N]$임과, $L_i = 1$인 여권과 $R_i = N$인 여권을 각각 발급받는것은 동치이다.

Observation 1

모든 도시를 방문하기 위하여 발급받아야 하는 여권의 최소 개수는 X_k번 도시에서 시작해서 $L_i = 1$인 여권과 $R_i = N$인 여권을 각각 발급받기 위하여 발급받아야 하는 여권의 최소 개수와 동치이다.

어떤 도시 x에서 여권을 발급받아, $[L_x, R_x]$에 속하는 도시 y로 이동하여 여권을 발급받는 행동을 x번 정점에서 y번 정점으로 이동하는 간선으로 생각하자. 따라서, 문제는 X_k번 도시에서 시작하여 $L_i = 1$인 도시 L과 $R_i = N$번 도시 R로 이동하는 경로를 찾는 것이다. 이 때 경로는 꼭 X_k에서 시작하여 L을 방문한 후 R를 방문할 필요 없이, 중간에 갈라져서 L로 가는 경로와 R로 가는 경로를 다르게 선택해도 된다. 물론, 이 때에도 사용한 정점의 총 개수는 중복하지 않고 센다.





위 그림처럼, 최적해는 X_k에서 시작하여 한 경로를 따라 임의의 정점 W까지 이동 후, W에서 갈라져 L과 R로 각각 이동하는 형태이다.

Observation 2

X_k에서 시작하여 정점 L과 R로 가는 최단경로의 형태는, 우선 임의의 정점 W로 이동한 후 갈라져 L과 R로 이동하는 것이다. 이 때 $X_k \to W$, $W \to L$, $W \to R$의 각 경로에는 겹치는 정점이 없다.

이제, 문제를 빠르게 해결하기 위하여 모든 간선의 방향을 뒤집고, 조건을 만족하는 L 정점들에서 다른 모든 정점들로의 최단경로 D_L, 조건을 만족하는 R 정점들에서 다른 모든 정점들로의 최단경로 D_R를 구하자. 이후 각 정점 w에서 기본 가중치 $D_L[w] + D_R[w]$로 시작하여 다른 정점들로 가는 최단경로 D를 구할 수 있다면, D가 바로 문제에서 요구하는 답이 됨을 알 수 있다.

모든 간선의 가중치가 1이고, 정점이 $O(N)$개, 간선은 $O(N^2)$개이니 BFS를 통해 $O(N^2)$에 문제를 해결할 수 있다.

CheckPoint

어떤 도시 x에서 여권을 발급받아, $[L_x, R_x]$에 속하는 도시 y로 이동하여 여권을 발급받는 행동을 x번 정점에서 y번 정점으로 이동하는 간선으로 생각할 때, Observation 1에 의해 이 그래프에서 X_k에서 시작하여 $L_i = 1$인 도시 L과 $R_i = N$번 도시 R로 이동하는 경로를 찾으면 된다. Observation 2에 의해 이러한 경로는 임의의 정점 W를 기준으로 $X_k \to W$, $W \to L$, $W \to R$ 3개의 최단 경로의 합을 구하는 문제로 생각할 수 있으니, BFS를 통해 $O(N^2)$에 문제를 해결할 수 있다.

Complexity

Time Complexity : $O(N^2)$

Subtask 5 (Full)

이제 실제로 N^2개의 간선을 이을 수 없는 상황인데, 간선의 형태가 구간에 있는 모든 정점에서 특정한 점을 연결하는 형태이므로, Segment Tree를 사용하여 최적화할 수 있다. 각 정점을 리프로 하는 Segment Tree를 dummy node로 만들고, Segment Tree의 루트로 방향을 부여하면 특정 구간에서 정점으로 간선을 이을 수 있다. 한 구간에서 정점으로 간선을 연결하는데 $O(logN)$개의 간선만 필요하니, 전체 그래프의 정점은 $O(N)$, 간선은 $O(NlogN)$개이다.

하지만 Segment Tree의 간선들은 가중치가 0이기 때문에 단순한 BFS를 사용할 수는 없다. Segment Tree에 속하는 간선들은 가중치가 0, 나머지 간선들은 모두 가중치가 1로 전체 그래프의 간선들의 가중치가 0 혹은 1이니 01-BFS를 사용하면 $O(NlogN)$에 문제를 해결할 수 있다.

CheckPoint

$O(N^2)$개의 간선을 Segment Tree를 사용하여 $O(NlogN)$개의 간선들로 압축할 수 있다. 이제, 이 그래프에서 가중치가 0 혹은 1이니 01-BFS를 사용하면 $O(NlogN)$에 문제를 해결할 수 있다.

Complexity

Time Complexity : $O(NlogN)$

코드

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

const int MAXN = 1e6;

int N, Q;
pii A[MAXN+10];
vector<int> adj[MAXN+10];

int init(int node, int tl, int tr)
{
    if(tl==tr) return tl;
    int mid=tl+tr>>1;
    adj[init(node*2, tl, mid)].push_back(node+N);
    adj[init(node*2+1, mid+1, tr)].push_back(node+N);
    return node+N;
}

void query(int node, int tl, int tr, int l, int r, int k)
{
    if(r<tl || tr<l) return;
    if(l<=tl && tr<=r)
    {
        if(tl==tr) adj[tl].push_back(k);
        else adj[node+N].push_back(k);
        return;
    }
    int mid=tl+tr>>1;
    query(node*2, tl, mid, l, r, k);
    query(node*2+1, mid+1, tr, l, r, k);
}

int D1[MAXN+10], D2[MAXN+10], D3[MAXN+10];

void bfs(int *D)
{
    deque<pii> Q;
    vector<pii> V;
    for(int i=1; i<=N; i++) if(D[i]) V.push_back({D[i], i});
    sort(V.begin(), V.end());
    for(int p=0; p<V.size();)
    {
        Q.push_back({V[p].second, V[p].first}); p++;
        while(!Q.empty())
        {
            pii t=Q.front(); Q.pop_front();
            if(D[t.first]!=t.second) continue;

            int now=t.first;
            for(; p<V.size() && V[p].first<=D[now]+1; p++) Q.push_back({V[p].second, V[p].first});
            for(auto nxt : adj[now])
            {
                if(D[nxt]!=0 && D[nxt]<=D[now]+1) continue;
                if(nxt<=N)
                {
                    D[nxt]=D[now]+1;
                    Q.push_back({nxt, D[nxt]});
                }
                else
                {
                    D[nxt]=D[now];
                    Q.push_front({nxt, D[nxt]});
                }
            }
        }
    }
    for(int i=1; i<=N; i++) if(!D[i]) D[i]=N+N;
}

int main()
{
    scanf("%d", &N);
    for(int i=1; i<=N; i++) scanf("%d%d", &A[i].first, &A[i].second);

    init(1, 1, N);
    for(int i=1; i<=N; i++)
    {
        auto [l, r]=A[i];
        query(1, 1, N, l, r, i);
    }

    for(int i=1; i<=N; i++) if(A[i].first==1) D1[i]=1;
    bfs(D1);

    for(int i=1; i<=N; i++) if(A[i].second==N) D2[i]=1;
    bfs(D2);

    for(int i=1; i<=N; i++) D3[i]=D1[i]+D2[i]-1;
```

```
    bfs(D3);

    scanf("%d", &Q);
    while(Q--)
    {
        int t;
        scanf("%d", &t);
        if(D3[t]>N) D3[t]=-1;
        printf("%d\n", D3[t]);
    }
}
```

2.2.2 APIO

2019

`math` `number theory` `~ solution`

APIO 19 P1 STRANGE DEVICE

문제

문제 링크

https://www.acmicpc.net/problem/17634
https://oj.uz/problem/view/APIO19_strange_device

문제 요약

$x = (t + \lfloor \frac{t}{B} \rfloor) \pmod A$, $y = t \pmod B$ 일 때, t에 대한 순서쌍 (x, y)를 생각하자.
t의 구간들 $[l_i, r_i]$이 N개 주어질 때, 적어도 하나의 구간에 포함되는 모든 t들에 대하여 가능한 서로 다른 순서쌍 (x, y)의 개수를 구하여라.

제한

- $N \le 10^6$
- $1 \le A, B \le 10^{18}$
- $0 \le l_i \le r_i \le 10^{18}$, $r_i < l_{i+1}$

입력 / 출력

> Input

$N\ A\ B$
$l_1\ r_1$
$l_2\ r_2$
$\vdots$
$l_N\ r_N$

> Output

ans

풀이

우선, y의 주기가 B임에 착안하여 t가 B씩 증가함에 따라 순서쌍 (x, y)가 어떻게 변하는지 관찰하자. $0 \le k < B$에 대하여,
$t = k, k + B, k + 2B, \cdots, k + nB$일 때 x는 다음과 같이 변한다.

t	$x \pmod A$
k	k
$k + B$	$k + B + 1$
$k + 2B$	$k + 2B + 2$
$\vdots$	$\vdots$
$k + nB$	$k + nB + n$

t가 B 증가할 때마다 x는 $(mod\ A)$ 속에서 $B+1$씩 증가한다는 사실을 관찰할 수 있다. 주기를 구하기 위해 $x_k \equiv x_{k+nB}\ (mod\ A)$인 최소 n을 구하자.

$$A \mid n(B+1),\ n = \frac{A}{gcd(A, B+1)}$$

위 결과를 통해 고정된 y에 대하여 주기가 $n = \frac{A}{gcd(A,B+1)}$이니, 가능한 B개의 모든 y에 대해서는 전체 $T = \frac{AB}{gcd(A,B+1)}$임을 알 수 있다.

Observation 1

(x, y)의 주기는 $T = \frac{AB}{gcd(A,B+1)}$이다.

이제 (x, y) 대신, $t\ (mod\ T)$로 대체하여 생각해도 된다. 만약 어떤 구간의 길이가 T 이상이면, 이 구간 하나만으로도 $0 \sim T-1$의 모든 수들을 다 포함하니, 답은 T이다. 모든 구간의 길이가 T 미만이면, 각 구간은 $0 \sim T-1$의 수들을 원형으로 배열한 모양에서 하나의 구간으로 대응된다. 0을 기준으로 원형 구간들을 모두 쪼개고, 전체 구간들을 정렬한 후 sweeping을 통해 $O(NlogN)$에 합집합의 크기를 구할 수 있다.

CheckPoint

Observation 1에서 (x, y)의 주기가 T이니 순서쌍 (x, y) 대신 $t\ (mod\ T)$로 대체할 수 있다.
구간 $[l_i, r_i]$는 $0 \sim T-1$의 수들을 원형으로 배열한 모양에서 하나의 구간으로 대응시킬 수 있으니, 0을 기준으로 원형 구간들을 선형 구간들로 쪼개고, 정렬한 후 sweeping을 통해 $O(NlogN)$에 합집합의 크기를 구할 수 있다.

마지막으로, $1 \le A, B \le 10^{18}$이니 $T = \frac{AB}{gcd(A,B+1)}$가 `long long` 자료형의 범위를 벗어날 수 있으므로, $min(T, 10^{18})$의 값을 사용하여 overflow를 방지할 수 있다.

Complexity

Time Complexity : $O(NlogN)$

코드

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

const int MAXN = 1e6;

int N;
ll A, B, L;
pll P[MAXN+10];

map<ll, ll> M;

void addrange(ll l, ll r)
{
    M[l]++;
    M[r+1]--;
}

int main()
{
    scanf("%d%lld%lld", &N, &A, &B);
    for(int i=1; i<=N; i++) scanf("%lld%lld", &P[i].first, &P[i].second);

    L=__gcd(A, B+1);

    if(A/L>=(ll)1e18/B) L=1e18;
    else L=A/L*B;

    for(int i=1; i<=N; i++)
    {
        ll l=P[i].first, r=P[i].second;
        if(r-l+1>=L) return !printf("%lld\n", L);
        if(l%L<=r%L) addrange(l%L, r%L);
        else
        {
            addrange(l%L, L-1);
            addrange(0, r%L);
        }
    }

    M[L];
    ll t=0, ans=0;
    for(auto it=M.begin(); it!=M.end(); it++)
    {
        if(it->first==L) break;
        t+=it->second;
        if(t) ans+=next(it)->first-it->first;
    }
    printf("%lld\n", ans);
}
```

2015

dp greedy ~ solution

APIO 15 P1 BALI SCULPTURES

문제

문제 링크

https://www.acmicpc.net/problem/10846
https://oj.uz/problem/view/APIO15_sculpture

문제 요약

길이 N의 수열 Y가 주어질 때, 이 수열을 X $(A \le X \le B)$개의 구간으로 쪼개어 각 구간별 Y_i의 합들의 bitwise OR을 최소화하여라.

제한

- $1 \le N \le 2{,}000$
- $1 \le A \le B \le N$
- $0 \le Y_i \le 10^9$

Subtask 4 : $1 \le N \le 100$, $1 \le A \le B \le N$
Subtask 5 : $1 \le N \le 2{,}000$, $1 = A \le B \le N$

입력 / 출력

> Input

N A B
Y_1 Y_2 ··· Y_N

> Output

ans

풀이

Subtask 4와 Subtask 5가 서로를 포함하지 않는, 두 개의 문제로 구성되어 있음에 유의하자.
Subtask 4의 경우, $1 \le N \le 100$이고, Subtask 5의 경우 $1 \le N \le 2{,}000$인 대신 $A = 1$로, 가능한 구간의 개수를 최소화 시켜야 한다.

"$P[i] := Y[i]$의 누적합 배열" 으로 정의한다.

Subtask 4

- $1 \le N \le 100$, $1 \le A \le B \le N$

생각할 수 있는 가장 직관적인 풀이는 다음과 같다.

> Wrong Solution

$dp[i][k] := 1 \sim i$까지의 수들을 k개의 구간으로 쪼갤 때 각 구간합들의 bitwise OR 값의 최솟값
$dp[i][k] = min_{j<i}(dp[j][k-1] | (P[i] - P[j]))$
위 정의와 점화식의 DP를 통해 $O(N^3)$에 답을 구한다.

하지만, 위 풀이는 올바른 답을 보장하지 않는다.
DP가 최적해를 보장하기 위해서는 전체 문제를 최적으로 풀기 위해서 부분문제 또한 최적으로 풀어야 한다는 **최적 부분 구조**가 성립해야 한다. 만약 작은 부분문제에서 가능한 답이 `10(2)`와 `01(2)`였고, 큰 전체문제로 가기 위해서는 이 답에 `10(2)`를 추가해야 된다고 가정하자. 그렇다면 전체문제에서의 값은 각각 `10(2)`, `11(2)`가 되며, 전체 문제에서의 최적해는 `10(2)`이다. 하지만 부분문제에서의 최적해는 `01(2)`로, 이 최적해로 전체문제를 풀면 `11(2)`로, 최적해를 얻을 수 없다. 따라서 문제의 구조는 최적 부분 구조가 성립하지 않음을 알 수 있다. 특히, 많은 경우에 bitwise 연산을 할 때 최적 부분 구조가 성립하지 않는다.

문제의 답을 최고 비트부터 하나씩 결정해가는 풀이를 생각하자.
최적해의 $T+1$이상의 비트를 다 결정했다고 가정하고 T번째 비트를 결정하기 위하여 다음의 DP를 생각하자.

> Definition 1

T를 최상위 비트에서부터 감소시키며, 최적해의 $T+1$이상의 비트를 다 결정했다고 가정하고, T번째 비트가 0이 될 수 있는지 확인하자.
$dp[i][k]:=1\sim i$까지의 수들을 k개의 구간으로 쪼갤 때 각 구간합들을 bitwise OR한 값들 중, $T+1$이상의 비트는 지금까지 구한 최적해와 똑같으며, T번째 비트는 0이 되도록 할 수 있는가? (`True` / `False`)

위와 같이 정의한 후, $dp[j][k-1](j<i)$에서 DP값을 받아올 때, $P[i]-P[j]$구간합을 bitwise OR함으로 인해 지금까지 구한 최적해의 상위 비트에 문제가 발생하지 않는지 확인하고, T번째 비트가 0이 될 수 있는지를 확인하여 transition을 한다.

위 DP의 경우, 가능/불가능을 따지고 있기 때문에 최적 부분 구조에 대한 문제가 발생하지 않는다. 또한, 최상위 비트부터 아래쪽으로, 최적해의 비트를 결정해 나가고 있는 방식으로 문제를 해결하니, 최적해를 무조건 찾을 수 있다. T번째 비트가 0이 될 수 있는지 확인하기 위해서는
$dp[N][A],dp[N][A+1],\cdots,dp[N][B]$중에 참인 것이 있는지 확인하면 된다.

$logX$개의 비트 각각을 $O(N^2)$개의 상태 하나를 $O(N)$번의 transition으로 문제를 해결하니, 전체 $O(N^3logX)$에 문제를 해결할 수 있다.

> CheckPoint

Definition 1과 같이 DP를 정의하면 최고 비트부터 하나씩 최적해를 결정할 수 있고, $O(N^3logX)$에 문제를 해결할 수 있다.

> Complexity

Time Complexity : $O(N^3logX)$

Subtask 5

- $1\le N\le 2{,}000,\ 1=A\le B\le N$

Subtask 5의 경우 가능한 구간의 개수를 최소화 시켜야 하는 대신 N의 크기가 늘어났다. 이제, $dp[i][k]$의 `True` / `False` DP 대신, k항을 DP값으로 변환하여 조건을 만족하기 위한 구간의 개수의 최솟값을 저장하자.

> Definition 2

T를 최상위 비트에서부터 감소시키며, 최적해의 $T+1$이상의 비트를 다 결정했다고 가정하고, T번째 비트가 0이 될 수 있는지 확인하자.
$dp[i]:=1\sim i$까지의 수들을 여러 구간으로 쪼갤 때 각 구간합들을 bitwise OR한 값들 중, $T+1$이상의 비트는 지금까지 구한 최적해와 똑같으며, T번째 비트는 0이 되도록 할 수 있는 구간의 최소 개수

transition은 Subtask 4의 경우와 거의 유사하다. 조건을 만족시키기 불가능하거나 필요한 구간의 개수가 B를 넘어가는 경우, $dp[i]$에 무한히 큰 값을 넣어 해결할 수 있다. T번째 비트가 0이 될 수 있는지 확인하기 위해서는 $dp[N]$이 B보다 작거나 같은지 확인하면 된다.

CheckPoint

구간의 개수를 최소화시키면 된다는 점을 이용하면 Definition 2와 같이 DP를 수정하면 최고 비트부터 하나씩 최적해를 결정할 수 있고, $O(N^2 logX)$에 문제를 해결할 수 있다.

Complexity

Time Complexity : $O(N^2 logX)$

코드

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

const int MAXN = 2000;

int N, A, B;
ll Y[MAXN+10], ans=0, dp1[MAXN+10], dp2[MAXN+10][MAXN+10];

int main()
{
    scanf("%d%d%d", &N, &A, &B);
    for(int i=1; i<=N; i++) scanf("%lld", &Y[i]), Y[i]+=Y[i-1];

    ans=(1ll<<46)-1;
    if(A==1)
    {
        for(int i=45; i>=0; i--)
        {
            ll now=1ll<<i;
            ans^=now;
            for(int j=1; j<=N; j++) dp1[j]=987654321; dp1[0]=0;
            for(int j=1; j<=N; j++) for(int k=0; k<j; k++) if((ans|(Y[j]-Y[k]))==ans) dp1[j]=min(dp1[j], dp1[k]+1);
            if(dp1[N]>B) ans|=now;
        }
    }
    else
    {
        for(int i=45; i>=0; i--)
        {
            ll now=1ll<<i;
            ans^=now;
            memset(dp2, 0, sizeof(dp2)); dp2[0][0]=1;
            for(int j=1; j<=N; j++) for(int k=1; k<=N; k++) for(int p=0; p<j; p++) if((ans|(Y[j]-Y[p]))==ans) dp2[j][k]|=dp2[p][k-1];
            bool flag=false;
            for(int j=A; j<=B; j++) flag|=dp2[N][j];
            if(!flag) ans|=now;
        }
    }
    printf("%lld", ans);
}
```

`sqrt_decomposition` `dijkstra` `graphs` `~ solution`

APIO 15 P2 JAKARTA SKYSCRAPERS

문제

문제 링크

https://www.acmicpc.net/problem/10847
https://oj.uz/problem/view/APIO15_skyscraper

문제 요약

0,1,⋯ ,$N-1$번의 N개의 빌딩이 있고, 0,1,⋯ ,$M-1$번의 M명의 전령들이 있다.
i번째 전령은 처음에 B_i번 빌딩에서 시작하고, 한 번의 점프로 정확히 P_i개의 빌딩을 건너뛰어 이동할 수 있다. (x번 빌딩에서 $x+P_i$ ($x+P_i<N$)번 빌딩으로 이동, $x-P_i$ ($x-P_i\geq 0$)번 빌딩으로 이동)
0번 전령이 1번 전령에게 소식을 전달하려고 하며, 소식을 전해 들은 전령은 현재 칸에 있는 다른 전령에게 소식을 전달하거나, 다른 빌딩으로 한번 점프를 할 수 있을 때 필요한 점프의 총 횟수의 최솟값을 구하여라.

제한

- $1\leq N\leq 30{,}000$
- $2\leq M\leq 30{,}000$
- $0\leq B_i<N$
- $1\leq P_i\leq 30{,}000$

입력 / 출력

> Input

$N\ M$
$B_0\ P_0$
$B_1\ P_1$
⋮
$B_{M-1}\ P_{M-1}$

> Output

ans

풀이

Subtask 3

- $N\leq 2{,}000$, $M\leq 2{,}000$

i번째 전령은 B_i번 빌딩에서 시작해서 B_i+kP_i ($0\leq B_i+kP_i<N$)번 빌딩으로 점프할 수 있으니, B_i번 정점에서 B_i+kP_i번 정점으로 가중치 $|k|$의 간선을 이어 준다. 이제, 문제는 B_0번 정점에서 B_1번 정점으로 가는 최단경로를 구하는 것이니 다익스트라 알고리즘을 사용하여 문제를 해결할 수 있다.

정점의 개수는 $O(N)$, 간선의 개수는 최악의 경우에 $O(NM)$개까지 가능하니, $O(ElogV)$ 다익스트라 알고리즘을 사용하면 $O(NMlogN)$, $O(E+V^2)$ 다익스트라 알고리즘을 사용하면 $O(NM+N^2)$에 문제를 해결할 수 있다.

CheckPoint

B_i번 정점에서 $B_i + kP_i$ $(0 \le B_i + kP_i < N)$번 정점으로 가중치 k의 간선을 이어 만든 정점 $O(N)$개, 간선 $O(NM)$인 그래프를 생각하자. 이 그래프에서 다익스트라 알고리즘으로 $O(N^2 + NM)$의 시간에 최단경로를 구하여 문제를 해결할 수 있다.

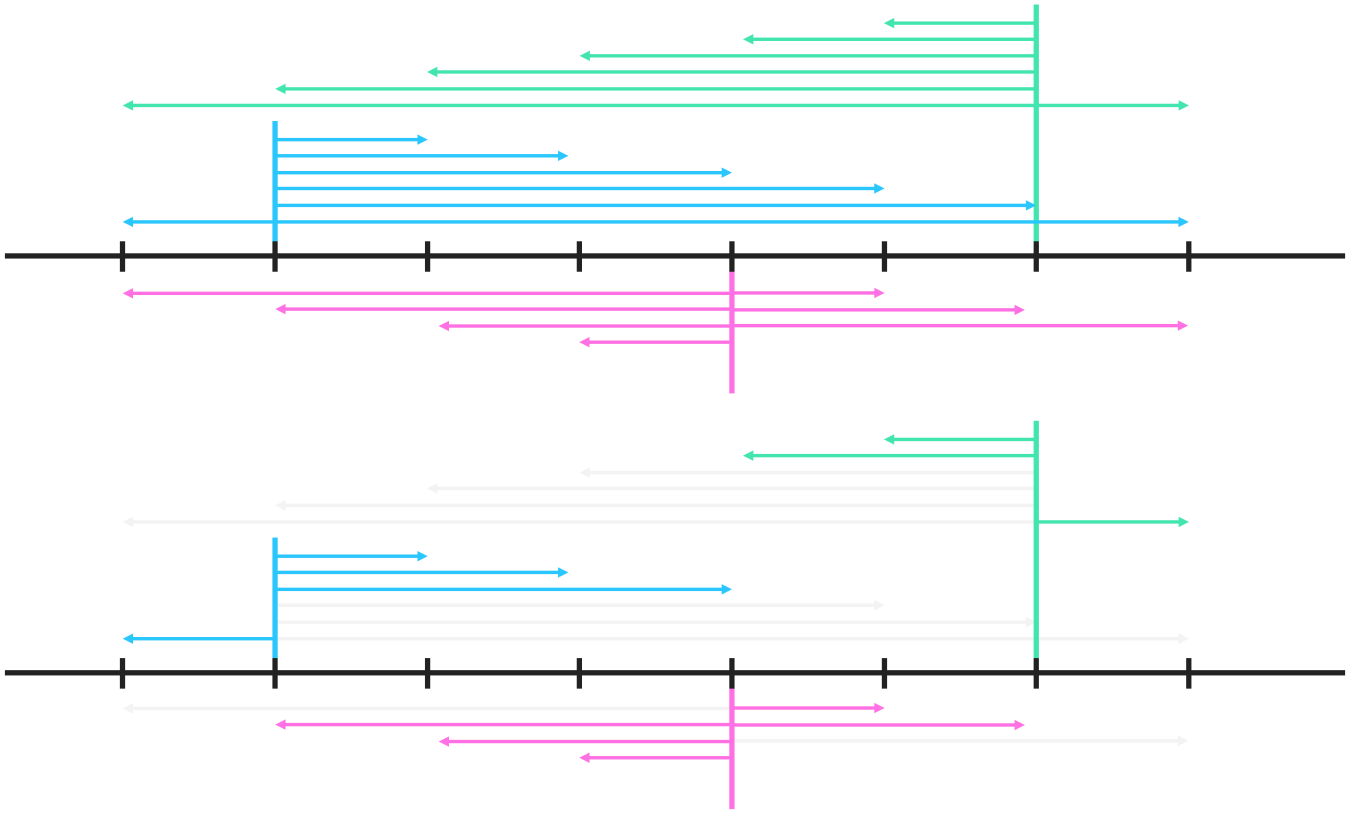
Complexity

Time Complexity : $O(NMlogN)$ or $O(N^2 + NM)$

Subtask 5 (Full)

위 그래프에서 불필요한 간선이 있는지 살펴보자.

만약 어떤 두 i, j가 $P_i = P_j$이며, $B_i \equiv B_j \pmod{P_i}$, $B_i < B_j$라고 가정하자. 그렇다면 i번 전령은 j번 전령의 오른쪽까지 이동할 일이 있을 때, j번 전령의 위치까지만 이동하고 j번 전령한테 소식을 전해도 된다. 같은 방법으로 j번 전령 또한 i번 전령을 넘어 왼쪽으로 이동할 필요가 없다. 따라서, i번 정점에서 나오는 간선은 j번 위치를 오른쪽으로 넘지 못하도록 하고 j번 정점에서 나오는 간선은 i번 정점을 왼쪽으로 넘지 못하도록 할 수 있다. 위와 같은 압축은 $P_i = P_j$이고 $B_i \equiv B_j \pmod{P_i}$인 모든 i, j들을 하나로 묶어, 같은 수의 간선을 사용해도 표현할 수 있도록 해준다.



Observation 1

어떤 두 i, j가 $P_i = P_j$이며, $B_i \equiv B_j \pmod{P_i}$, $B_i < B_j$일 때, i번 전령이 오른쪽으로 j번 전령을 넘지 않고, j번 전령이 왼쪽으로 i번 전령을 넘지 않도록 이동하는 최적해가 존재한다.

이러한 압축은 간선의 개수를 얼마나 효과적으로 줄여줄 수 있을까?

P와 P로 나눈 나머지인 k를 고정하고 나면, $P_i = P$, $B_i \equiv k \pmod{P}$인 모든 i들을 그래프로 표현하기 위해서는 $O(\frac{N}{P_i})$개의 간선이 필요하다. 또한, k는 P로 나눈 나머지이므로 서로 다른 k는 최대 P개 가능함을 알 수 있다.

이제, 최악의 경우에는 각 전령들이 $P_i = 1$인 것이 1개, $P_i = 2$인 것이 2개, ..., $P_i = \sqrt{M}$인 것이 $\sqrt{M}$개와 같이 있을 때이다. $P_i = 1$인 것들의 간선의 개수의 합은 $\frac{N}{1} \cdot 1$, $P_i = 2$인 것들은 $\frac{N}{2} \cdot 2$, ..., $P_i = \sqrt{M}$인 것들은 $\frac{N}{\sqrt{M}} \cdot \sqrt{M}$로, 전체 합은 다음과 같다.

$$\sum_{1 \le p \le \sqrt{M}} \frac{N}{p} \cdot p = O(N\sqrt{M})$$

Observation 2

$P_i = P_j$이고 $B_i \equiv B_j \pmod{P_i}$인 모든 i, j들을 하나로 묶어 불필요한 간선들을 제거하면, 압축된 그래프에서 사용되는 전체 간선의 개수는 $O(N\sqrt{M})$개이다.

이제, 위와 같이 만든 압축된 그래프는 $O(N)$개의 정점과 $O(N\sqrt{M})$개의 간선들로 구성되어 있고, 다익스트라 알고리즘을 활용하면 $O(N\sqrt{M}logN)$에 문제를 해결할 수 있다.

CheckPoint

Observation 1에 의해 Observation 2와 같이 그래프를 압축할 수 있고, 정점 $O(N)$, 간선 $O(N\sqrt{M})$개의 압축된 그래프에서 다익스트라 알고리즘으로 $O(N\sqrt{M}logN)$의 시간에 최단경로를 구하여 문제를 해결할 수 있다.

구현할 때, 다익스트라의 구현이 충분히 빠르지 않으면 `AC`를 받기 힘드니 빠른 다익스트라의 구현을 사용할 수 있도록 주의하자.

Complexity

Time Complexity : $O(N\sqrt{M}logN)$

코드

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

const int MAXN = 3e4;
const int INF = 1e9;

int N, M, S, E;
vector<int> A[MAXN+10];
vector<pii> adj[MAXN+10];
int dist[MAXN+10];

struct Queue
{
    int u, w;
    bool operator < (const Queue &p) const { return w>p.w; }
};

int main()
{
    scanf("%d%d", &N, &M);
    for(int i=1; i<=M; i++)
    {
        int b, p;
        scanf("%d%d", &b, &p);
        if(i==1) S=b;
        if(i==2) E=b;
        A[p].push_back(b);
    }

    for(int i=1; i<=MAXN; i++)
    {
        if(A[i].empty()) continue;
        vector<vector<int>> V;
        vector<int> comp;
        V.resize(i);
        for(auto it : A[i]) V[it%i].push_back(it);
        for(auto it : A[i]) comp.push_back(it%i);

        sort(comp.begin(), comp.end());
        comp.erase(unique(comp.begin(), comp.end()), comp.end());

        for(auto it : comp)
        {
            sort(V[it].begin(), V[it].end());
            for(int j=1; j<V[it].size(); j++)
            {
                int now=V[it][j], bef=V[it][j-1];
                for(int k=bef+i; k<=now; k+=i) adj[bef].push_back({k, (k-bef)/i});
                for(int k=now-i; k>=bef; k-=i) adj[now].push_back({k, (now-k)/i});
            }
            for(int k=V[it].back()+i; k<N; k+=i) adj[V[it].back()].push_back({k, (k-V[it].back())/i});
            for(int k=V[it][0]-i; k>=0; k-=i) adj[V[it][0]].push_back({k, (V[it][0]-k)/i});
        }
    }

    priority_queue<Queue> PQ;
    for(int i=0; i<N; i++) dist[i]=INF;
    PQ.push({S, 0}); dist[S]=0;

    while(!PQ.empty())
    {
        Queue now=PQ.top(); PQ.pop();
        if(dist[now.u]<now.w) continue;
        for(auto nxt : adj[now.u]) if(dist[nxt.first]>now.w+nxt.second) PQ.push({nxt.first, now.w+nxt.second}), dist[nxt.first]=now.w+nxt.second;
    }
    if(dist[E]==INF) dist[E]=-1;
    printf("%d\n", dist[E]);
}
```

3. 알고리즘 & 자료구조 정리

3.1 알고리즘 & 자료구조 정리

4. 아이디어 & 테크닉 모음

4.1 아이디어 & 테크닉 모음

5. Study

5.1 Study

5.2 Random Platinum Defence

Random Problem

Date	Problem	Time Without Computer	Time With Computer	Failed Attempts	Total Penalty
2023-05-01	14961	12	15	2	67
2023-05-02	14756	11	23	3	94
2023-05-03	10266	2	16	1	38
2023-05-04	8916	2	8	0	10
2023-05-05	3946	12	14	0	26

6. Tags

6.1 0_1_bfs

- JOISC 2023 P1.3 Passport

6.2 binary_search

- JOISC 2023 P1.1 Two Currencies

6.3 data_structures

- JOISC 2023 P1.1 Two Currencies
- JOISC 2023 P1.3 Passport

6.4 dijkstra

- APIO 15 P2 Jakarta Skyscrapers
- JOISC 2023 P1.3 Passport

6.5 dp

- APIO 15 P1 Bali Sculptures

6.6 euler_tour_technique

- JOISC 2023 P1.1 Two Currencies

6.7 graphs

- APIO 15 P2 Jakarta Skyscrapers
- JOISC 2023 P1.3 Passport

6.8 greedy

- APIO 15 P1 Bali Sculptures

6.9 lca

- JOISC 2023 P1.1 Two Currencies

6.10 math

- APIO 19 P1 Strange Device

6.11 number theory

- APIO 19 P1 Strange Device

6.12 offline_queries

- JOISC 2023 P1.1 Two Currencies

6.13 pbs

- JOISC 2023 P1.1 Two Currencies

6.14 pst

- JOISC 2023 P1.1 Two Currencies

6.15 segtree

- JOISC 2023 P1.1 Two Currencies
- JOISC 2023 P1.3 Passport

6.16 sqrt_decomposition

- APIO 15 P2 Jakarta Skyscrapers

6.17 trees

- JOISC 2023 P1.1 Two Currencies

6.18 ~ solution

- _
- APIO 15 P1 Bali Sculptures
- APIO 15 P2 Jakarta Skyscrapers
- APIO 19 P1 Strange Device
- APIO 19 P3 Street Lamps
- JOISC 2023 P1.1 Two Currencies
- JOISC 2023 P1.3 Passport